RBAC访问控制实验

> *18307130089 吴嘉琪*

设计思路：

一.总体架构

使用python编程，模拟实现了基于RBAC模型的权限管理架构。

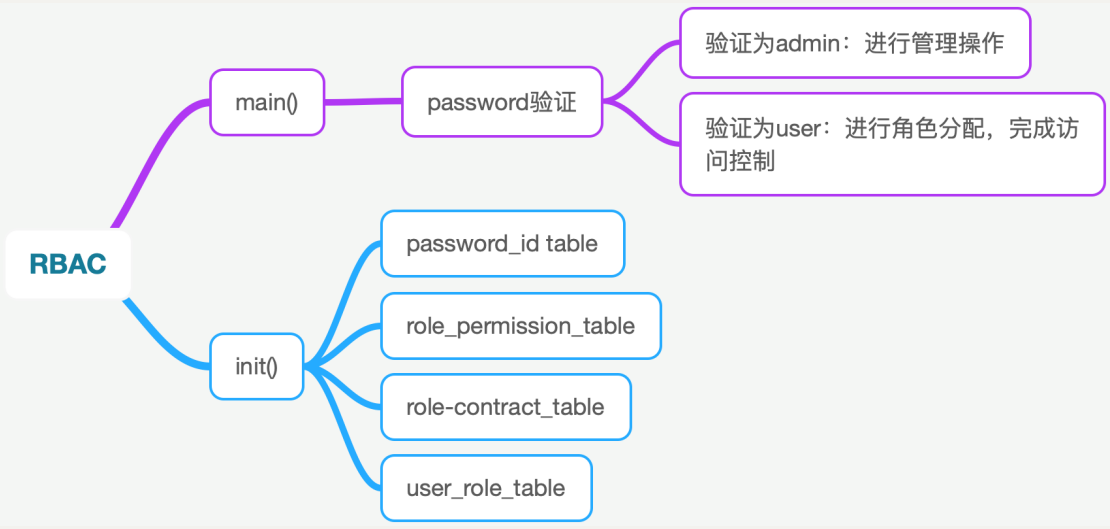
代码主要分为：**规则初始化（init.py）** 和 **主要操作(main.py)** 两大板块

init.py负责生成用户角色关联表、角色权限关联表、用户密码表等关键信息，并且以文件形式存储（存为npy格式）

main.py 分用户登陆和管理员登陆两大功能。

用户登陆，会进入RBAC模型进行权限验证和操作；

管理员登陆，可以对init中的各种表格和映射关系进行修改。



二.具体设计

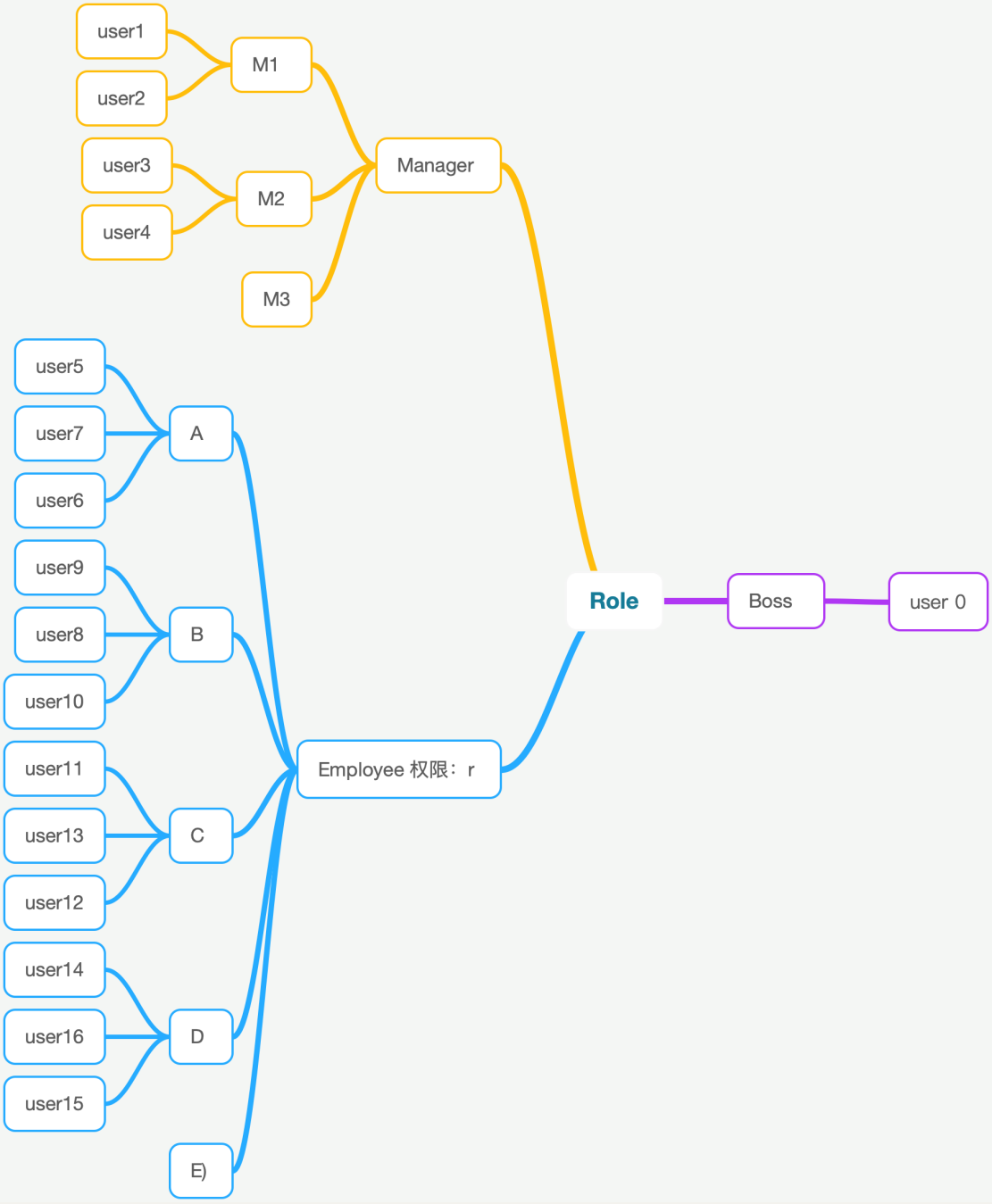
init部分

假设公司共有18名成员--user0--user16，和一位admin。

Password-id table:用于在第一次登陆时确认他们的身份。简单起见，用户密码表按如下初始化：

用户	密码
admin	ad
User0	0
User1	1
User2	2
User3	3
User4	4
User5	5
User6	6
User7	7
User8	8
User9	9
User10	10
User11	11
User12	12
User13	13
User14	14
User15	15
User16	16

User-role-table:存储用户角色的分配关系,分配关系如下：



role_contract_table:记录每个角色被分配的用户数量。在进行角色用户映射关系修改时，会被查找并且进行互斥检查。互斥检查依据：**boss角色仅能分配给1名用户；M1/M2角色最多仅能分配给2用户；员工不限。**

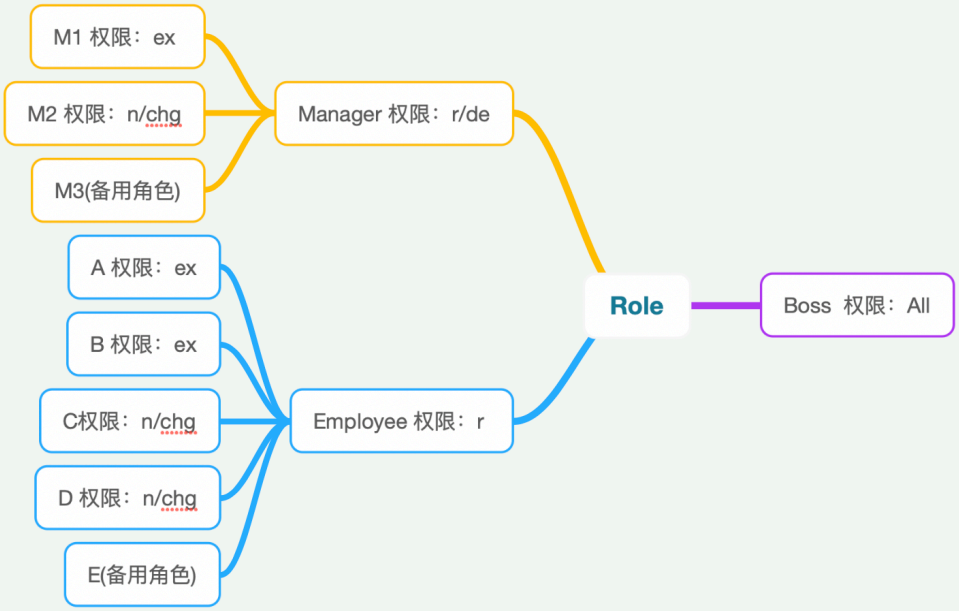
```
role_contract={'boss':1,'M1':2,'M3':2,'A':3,'B':3,'C':3,'D':3}
```

role_contract_table_standard:初始化时将role_contract_table复制一份，作为互斥标准；**修改映射关系后，boss和manager的用户数量不得超过此表**

User-permission-table:记录每个角色的权限。

具体权限参考说明文档：

> *公司员工A，B可以对其中的文件具有读取，执行的权限；他们的部门经理M1具有读取、删除、执行的权限；公司员工C、D具有这些文件的新建、读取、更改的权限；他们的经理M2具有新建、删除、读取、更改的权限；公司的大boss拥有所有权限。*




```
role_perm_tb=({'boss':['r','w','de','ex','n','chg'],'manager':
['r','de'],'M1':['ex'],'M2':['n','chg'],'employee':['r'],'A':
['ex'],'B':['ex'],'C':['n','chg'],'D':['n','chg']})
```

具体代码：

```
paswd_id={}
user_role_tb=list(''for i in range(100))
role_perm_tb={}
#role_extend_tb={}
role_contract={}
role_contract_standard={}


def init():

  #密码--用户表
  paswd_id={'admin':'ad','number i':'user i'}

  #用户--角色表 假定初始有17个user

  user_role_tb[0]='boss'
  user_role_tb[1]='M1'
```

```
  user_role_tb[2]='M1'
  user_role_tb[3]='M2'
  user_role_tb[4]='M2'
  for i in range(3):
    user_role_tb[5+i]='A'
  for i in range(3):
    user_role_tb[8+i]='B'
  for i in range(3):
    user_role_tb[11+i]='C'
  for i in range(3):
    user_role_tb[14+i]='D'


  #角色--权限表
  role_perm_tb=({'boss':['r','w','de','ex','n','chg'],'manager':
['r','de'],'M1':['ex'],'M2':['n','chg'],'employee':['r'],'A':
['ex'],'B':['ex'],'C':['n','chg'],'D':['n','chg']})

  #角色--互斥表，表面每个角色已经被分配给了多少用户
  role_contract={'boss':1,'M1':2,'M3':2,'A':3,'B':3,'C':3,'D':3}
role_contract_standard=role_contract

  np.save('role_perm_tb.npy',role_perm_tb)
  np.save('user_role_tb.npy',user_role_tb)
  np.save('paswd_id.npy',paswd_id)
  np.save('role_contract.npy',role_contract)
  np.save('role_contract_standard.npy',role_contract_standard)

init()
```

---

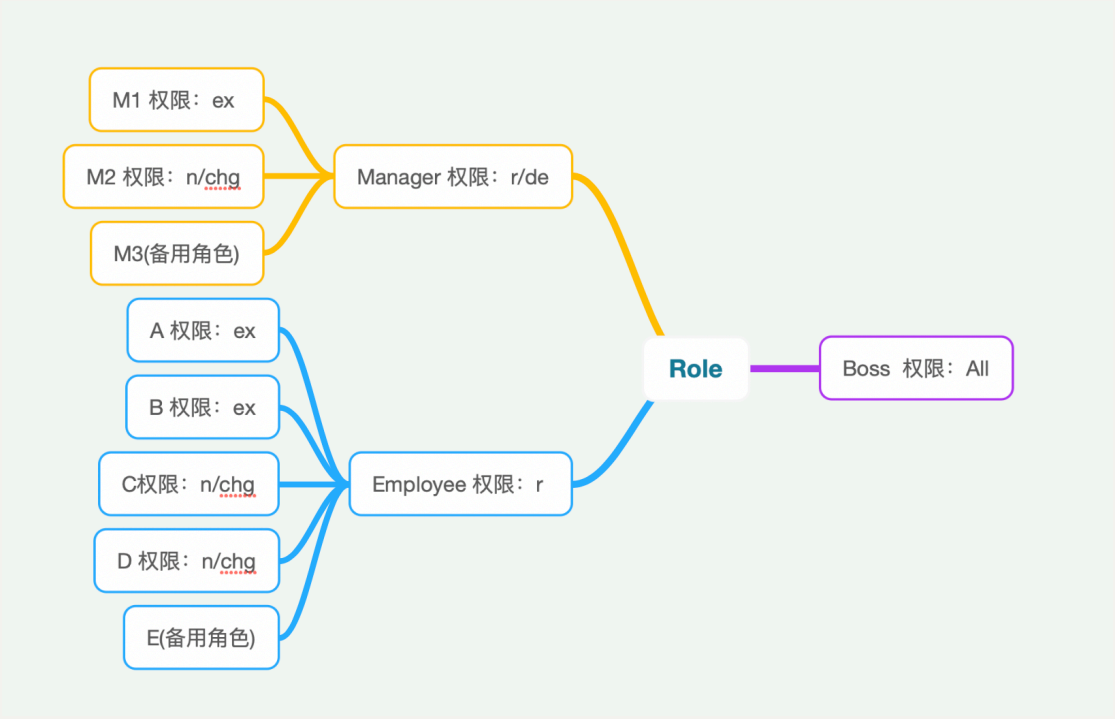
main 部分--角色结构与权限分配

使用python class，定义并完成角色之间的继承关系：

set（）函数用来初始化各角色的权限；

子类继承了父类的权限，并且拥有自己独有的权限；

角色权限继承关系：



具体代码:

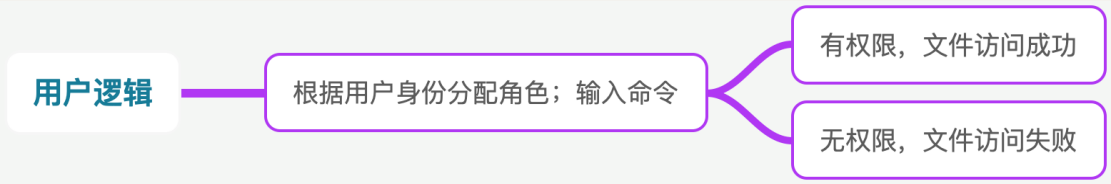
```
class role(object):
  def __init__(self):
    self.info =''
    self.perm =[]

  def set(self):
    #(self.num)++
    pass


class boss(role):
  def set(self):
    self.perm.append(role_perm_tb['boss'])

class manager(role):
  def set(self):
    self.perm.append(role_perm_tb['manager'])


class employee(role):
  def set(self):
    super().set()
    self.perm.append(role_perm_tb['employee'])
```

```
class A(employee):
  def set(self):
    super().set()
    self.perm.append(role_perm_tb['A'])

class B(employee):
  def set(self):
    super().set()
    self.perm.append(role_perm_tb['B'])

class C(employee):
  def set(self):
    super().set()
    self.perm.append(role_perm_tb['C'])

class D(employee):
  def set(self):
    super().set()
    self.perm.append(role_perm_tb['D'])

#备用
class E(employee):
  def set(self):
    super().set()
    self.perm.append(role_perm_tb['E'])

class M1(manager):
  def set(self):
    super().set()
    self.perm.append(role_perm_tb['M1'])

class M2(manager):
  def set(self):
    super().set()
    self.perm.append(role_perm_tb['M2'])

#备用
class M3(manager):
  def set(self):
    super().set()
    self.perm.append(role_perm_tb['M3'])
```

main 部分--用户逻辑：角色分配与文件访问

main函数具体框架：

```
def main():
  load()
#从外部加载init函数初始化的各张表格
  pwd=input('password:')
  print('identity:')
#根据密码判断用户类型
  if pwd == (paswd_id['admin']):
    print('admin')
    admin()#进入管理员逻辑
  else:
    print('normal user')
    user(pwd)#进入用户逻辑
```



进入用户逻辑后，首先查找user_role_tb，为用户分配角色；按角色新建并初始化相应的角色类对象；之后根据类对象的perm属性与用户输入的命令比对，权限匹配就成功完成文件访问操作；失败就返回失败信息

```
def user(pwd):
  print('------------------------')
  index=int(pwd)
  role=user_role_tb[index]
  print('your role:'+role)

  #角色新建并初始化相应的角色类对象
  if role=='boss':
    r=boss()
    r.set()
    perm=numpy_concatenate(r.perm)
    print(perm)

  elif role=='M1':
    r=M1()
    r.set()
```

```
    perm=numpy_concatenate(r.perm)
    print(perm)

  elif role=='M2':
    r=M2()
    r.set()
    perm=numpy_concatenate(r.perm)
    print(perm)

  elif role=='A':
    r=A()
    r.set()
    perm=numpy_concatenate(r.perm)
    print(perm)

  elif role=='B':
    r=B()
    r.set()
    perm=numpy_concatenate(r.perm)
    print(perm)

  elif role=='C':
    r=C()
    r.set()
    perm=numpy_concatenate(r.perm)
    print(perm)

  elif role=='D':
    r=D()
    r.set()
    perm=numpy_concatenate(r.perm)
    print(perm)

  print('------------------------')
  print('please type in your order:')

  while(1):
    order=input('>')
    if not order:
      break
  #根据类对象的perm属性与用户输入的命令比对
    elif order in perm:
      success() #成功

    else:
      fail()  #失败
```

操作效果：

```
(base) ➜  RBAC python init.py
(base) ➜  RBAC python main.py
password:0
identity:
normal user
----------------------
|   your role:boss
----------------------
|  please type in your order:
>w
you have done the acess succesfully!
>
```

用户0登陆，分配角色为boss；能执行所有权限操作；

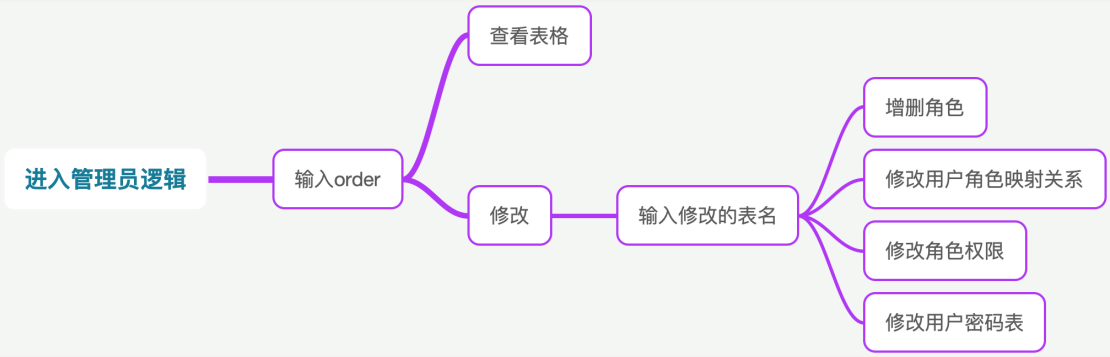
```
(base) ➜  RBAC python main.py
password:4
identity:
normal user
----------------------
|   your role:M2
----------------------
|  please type in your order:
>n
you have done the acess succesfully!
>de
you have done the acess succesfully!
>ex
sorry,you have no permission!
>
```

用户4登陆，角色为M2；对文件拥有新建、删除权限，但没有执行权限；

```
(base) ➜  RBAC python main.py
password:7
identity:
normal user
----------------------
|   your role:A
----------------------
|  please type in your order:
>w
sorry,you have no permission!
>
```

用户7登陆，角色为A，对文件没有写的权限；

main 部分--管理员逻辑：



进入管理员逻辑后，admin可以输入指令：

```
(base) ➜  RBAC python main.py
password:ad
identity:
admin
**********************************************************
*  You are the admin. Now you can type these orders: *
*   tables -- check all the tables                    *
*   modify --modify any tables                        *
**********************************************************
input orders:
```

输入命令tables，可以查看所有表格；

```
**********************************************************
input orders:tables
password_id table:
admin:ad
number i:user i
----------------------
role_permission_table:
boss:['r', 'w', 'de', 'ex', 'n', 'chg']
manager:['r', 'de']
M1:['ex']
M2:['n', 'chg']
employee:['r']
A:['ex']
B:['ex']
C:['n', 'chg']
D:['n', 'chg']
----------------------
user_role_table:
user 0: boss
user 1: M1
user 2: M1
user 3: M2
user 4: M2
user 5: A
user 6: A
user 7: A
user 8: B
user 9: B
user 10: B
user 11: C
user 12: C
user 13: C
user 14: D
user 15: D
user 16: D
----------------------
```

输入命令modify，可以选择修改 password-user、role-permission、user-role三大映射关系；

```
------------------------
input orders:modify
what to modify?
 type 1:password_id
 type 2:role_permission
 type 3:user_role
>
```

> *(因为用户密码与RBAC关联不大，因此此部分修改逻辑忽略)*

修改**role-permission**：例如修改M2的权限，可以选择增加权限或者删除权限；

```
----------------------
input orders:modify
what to modify?
 type 1:password_id
 type 2:role_permission
 type 3:user_role
>2
the role you want to modify is:
role:M2
type a:add permission
type b:delete permission
>>
```

增加ex权限：

```
type a:add permission
type b:delete permission
>>a
the permission to add is:
>>>ex
boss:['r', 'w', 'de', 'ex', 'n', 'chg']
manager:['r', 'de']
M1:['ex']
M2:['n', 'chg', 'ex']
employee:['r']
A:['ex']
B:['ex']
C:['n', 'chg']
D:['n', 'chg']
success
```

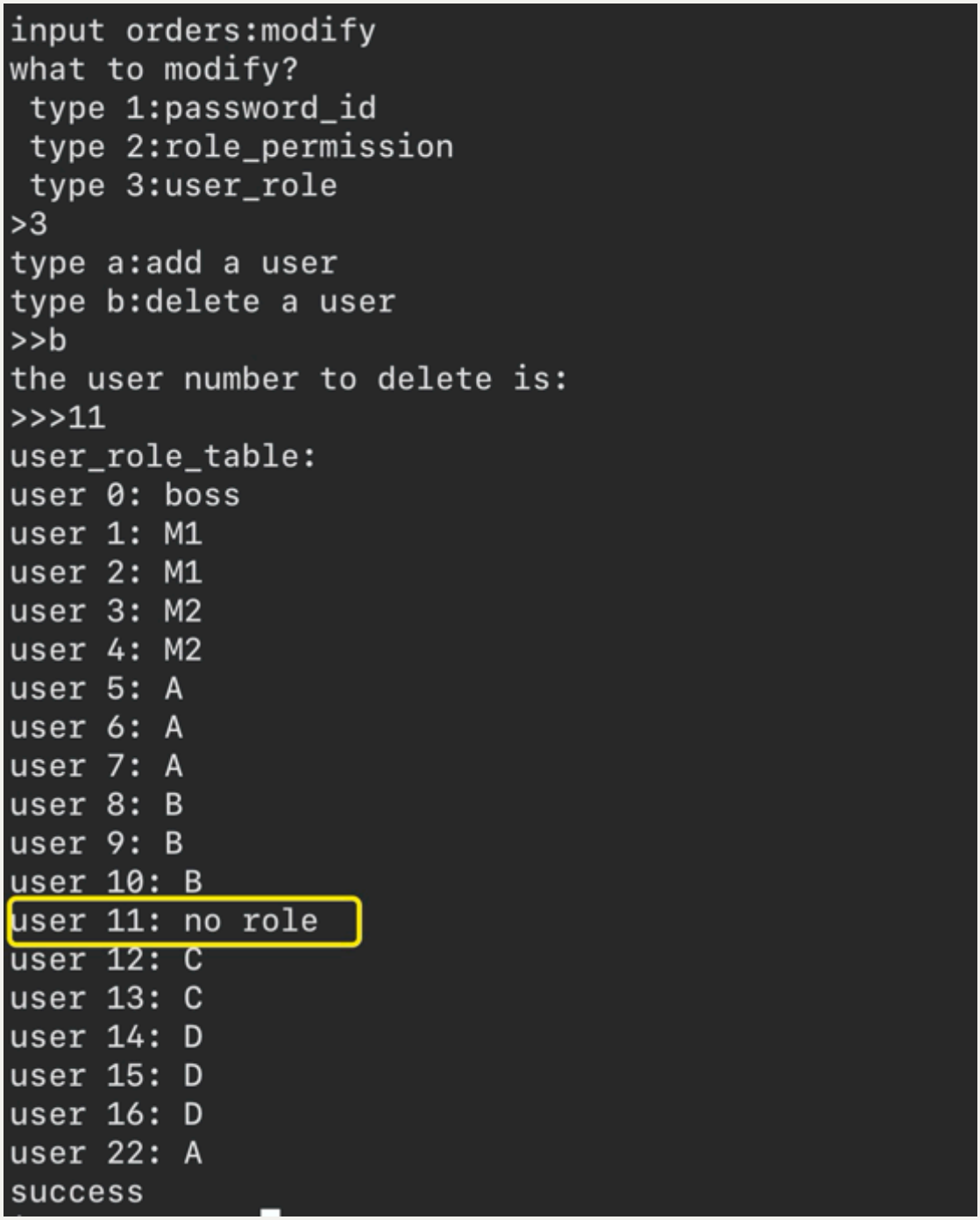
删除chg权限：

```
the role you want to modify is:
role:M2
type a:add permission
type b:delete permission
>>b
the permission to delete is:
>>>chg
boss:['r', 'w', 'de', 'ex', 'n', 'chg']
manager:['r', 'de']
M1:['ex']
M2:['n', 'ex']
employee:['r']
A:['ex']
B:['ex']
C:['n', 'chg']
D:['n', 'chg']
success
```

修改后，角色权限表更新后会保存近外部的npy文件中。

修改user-role映射关系：首先选择添加或删除user；

如果删除user，输入user编号；该user就会被解除角色绑定；



如果添加user，输入要赋予它的角色；

如果该角色是boss/manager，（初始化用户数<3）,就会进入互斥检查；如果现在已经绑定的用户数量已经大于等于标准，就不允许该操作：

```
if f=='a':
          print('the role you want to modify is:')
          r=input('role:')
          if role_contract_standard[r]<3 and
role_contract[r]>=role_contract_standard[r]:
            print('access denied!')
            break
```

举例：M2已经被绑定了两个用户，无法新增用户关系：

```
(base) ➜  RBAC python main.py
password:ad
identity:
admin
**********************************************************
*  You are the admin. Now you can type these orders: *
*   tables -- check all the tables
*   modify --modify any tables
**********************************************************
input orders:modify
what to modify?
 type 1:password_id
 type 2:role_permission
 type 3:user_role
>3
type a:add a user
type b:delete a user
>>a
the role you want to modify is:
role:M2
access denied!
```

如果互斥检查通过，该用户就被成功绑定了这个角色

```
*****************************************************
input orders:modify
what to modify?
 type 1:password_id
 type 2:role_permission
 type 3:user_role
>3
type a:add a user
type b:delete a user
>>a
the role you want to modify is:
role:A
the user number to add is:
>>>22
user 0: boss
user 1: M1
user 2: M1
user 3: M2
user 4: M2
user 5: A
user 6: A
user 7: A
user 8: B
user 9: B
user 10: B
user 11: C
user 12: C
user 13: C
user 14: D
user 15: D
user 16: D
user 22: A
success
```

修改后，用户角色表更新后会保存近外部的npy文件中。

具体代码：

```
def admin():
  print('*****************************************************')
  print('*  You are the admin. Now you can type these orders: *')
  print('*   tables -- check all the tables          *')
  print('*   modify --modify any tables            *')
  print('*****************************************************')

  global paswd_id
  global role_perm_tb
  global user_role_tb
  global role_contract
  global role_contract_standard

  #print(role_contract_standard)
```

```
  while (1):
    func=input('input orders:')
    if not func:
      break

    #查看表格
    elif func=='tables':
      print ('password_id table:')
      for key,value in paswd_id.items():
        print('{key}:{value}'.format(key = key, value = value))
      print('-----------------------')

      print('role_permission_table:')
      for key,value in role_perm_tb.items():
        print('{key}:{value}'.format(key = key, value = value))
      print('-----------------------')

      print('user_role_table:')
      for i in range(len(user_role_tb)):
        if user_role_tb[i]:
          print('user '+str(i)+': '+str(user_role_tb[i]))
      print('-----------------------')


    elif func=='modify':
      print ('what to modify?\n type 1:password_id\n type
2:role_permission\n type 3:user_role')
      md=input('>')
      if not func:
        break
      #switch(md):
      if md==('1'):
        pass

      elif md==('2'):
        print('the role you want to modify is:')
        r=input('role:')
        print('type a:add permission\ntype b:delete permission')
        f=input('>>')
        if f=='a':
          print('the permission to add is:')
          p=input('>>>')
          role_perm_tb[r].append(p)

        elif f=='b':
```

```
            print('the permission to delete is:')
            p=input('>>>')
            plist=(role_perm_tb[r])
            plist.remove(p)
            role_perm_tb[r]=plist

        #np.save('user_role_tb.npy',user_role_tb)
          for key,value in role_perm_tb.items():
            print('{key}:{value}'.format(key = key, value = value))
          print('success')

          #np.save('user_role_tb.npy',user_role_tb)

        elif md==('3'):

          print('type a:add a user\ntype b:delete a user')
          f=input('>>')
          if f=='a':
            print('the role you want to modify is:')
            r=input('role:')
            if role_contract_standard[r]<3 and
role_contract[r]>=role_contract_standard[r]:
              print('access denied!')
              break

            print('the user number to add is:')
            p=int(input('>>>'))
            user_role_tb[p]=r
            role_contract[r]+=1

            for i in range(len(user_role_tb)):
              if user_role_tb[i]:
                print('user '+str(i)+': '+str(user_role_tb[i]))

            np.save('user_role_tb.npy',user_role_tb)
            print('success')


          elif f=='b':
            print('the user number to delete is:')
            p=int(input('>>>'))
            user_role_tb[p]='no role'
            role_contract[r]-=1

            print('user_role_table:')
            for i in range(len(user_role_tb)):
```

```
            if user_role_tb[i]:
              print('user '+str(i)+': '+str(user_role_tb[i]))

        np.save('user_role_tb.npy',user_role_tb)
        print('success')
```

理解与总结

RBAC模型具体种类、实现方式多种多样，但最核心的思想是在用户与权限之间加一层角色的抽象，让权限只需要与角色关联；并且维护好用户--角色的映射关系，角色之间的分级与继承关系。最终目标是实现简洁有效的管理。

而需要解决的最大问题就是用户角色的分配问题，和不同用户竞争同一角色的互斥问题，最后是对这些庞大映射关系的管理问题。

在这次试验中我采用的方法是设置管理员和互斥标准表，映射关系在设计好后自动初始化并且妥善保存，之后对映射关系的修改、新老用户的流动，都需要经过管理员的修改与认证。